Optimizing the Robotics Closet

Sasha Krassovsky, John Lim, Andrey Ryabtsev

November 15, 2018

1 Introduction

The Husky Robotics team is a team of both undergraduate and graduate students from UW building a Mars rover to compete in the University Rover Challenge. The rover must perform a variety of tasks, such as autonomous navigation, soil collection, and typing on a keyboard with a robotic arm. The team is divided into subsystems: Chassis, Arm, Science, Electronics, Software, and Manufacturing. The subsystems work independently but collaborate to create the robot. Given all these tasks and the relatively short amount of time given to build out the functionality, efficiency is key when it comes to the build process.

The robotics team has only a small closet in the Mechanical Engineering Building. The closet stores the robot along with all of the tools, electronics, sheet metal, carbon fiber, and plastics that go into building the robot. Each item belongs to one of the subsystems, and is usually stored near other items belonging to that subsystem. Unfortunately, the robotics closet is not particularly organized, and members often spend on the order of 10 minutes looking for the thing they needed. Our goal is to find a reorganization plan for the items in the closet such that expected duration of each closet visit is minimized and items belonging to the same subsystem are in the same connected component on the shelves.

The closet has two shelves, one on each side. One side stores items in uniformly-sized boxes while the other side has several other classes of items, such as soldering irons, and boxes of various sizes. The rover is stored in the middle of the room on a big cart. Behind it is the robotic arm and the vscience station. In the back are the sheet metal, various metallic cylinders, wood, and rolls of carbon fiber. By examining the frequencies of retrieval by each team and treating the closet as a grid, we should be able to produce a better layout of the closet.

The robotics closet before optimization.

2 Related Work

Deepesh Singh's *A Beginner's guide to Shelf Space Optimization using Linear Programming* explains an integer programming formulation for finding the optimal layout of a store that maximizes sales. The guide organizes the store into several racks, each with a certain number of shelves. One key assumption is that only one product can fit on a rack. The model uses a table showing how many sales are generated by each item depending on which rack it is placed. The objective function is then the profit generated by a given arrangement. Arrangements are represented by a matrix, with each a_{ij} representing whether product j is on shelf i. If it is, the a_{ij} is 1 and 0 otherwise. The guide shows how to maximize this profit function. Although this approach is a good start, our problem differs in several ways: we can fit more than one box on each shelf, and our goal is to minimize time of retrieval rather than maximize profits.

TODO: More related work

3 Assumptions

Most items in the robotics closet are in boxes. As a result, we assume that all items are rectangular prisms. Although items such as the chemicals or soldering irons are not strictly rectangle-shaped, it would be strange for another item reached into the empty areas of the items' bounding boxes. Taking advantage of this, we approximate all items as being rectangular prisms, and measure their bounding boxes.

TODO: Fill in the rest of the assumptions

4 Model

We collected data from the robotics closet by measuring each item and shelf. Each shelf s is represented as an ordered triple (d, a, c), where d is the distance from the door in "shelf-lengths", a is the area of the shelf, and c is the connected components to which the self belongs.

Each item is represented with an ordered triple (a, p, s), where a is the area of the item, p is the probability that the item is selected, and s is the item's subsystem. Notice that we represent each item as a 2-dimensional area. This is valid because we assumed that each shelf had a single unit of depth. As a preprocessing step, we choose to minimize the area. However, we cannot rotate always the item into a horizontal orientation. Therefore, h is constant. For an item i with length l_i, width w_i, and height h_i, its area a_i is given by

$$a_i = \min(l_i, w_i) \cdot h_i$$

Let there be n shelves and m items, k subsystems, and l is the number of connected components. Our decision variables lie in two matrices

$$W = (w_{ij}), 1 \leq i \leq n, 1 \leq j \leq m$$

$$C = (c_{ij}), 1 \leq i \leq l, 1 \leq j \leq k$$

Each w_{ij} and c_{ij} is a binary variable, i.e. each $w_{ij}, c_{ij} \in 0, 1$. Each w_{ij} is 0 if item j is not on shelf i and 1 otherwise. c_{ij} is 1 if subsystem j lies in connected component i and 0 otherwise. C is a $l \times k$ matrix because there are l connected components and k subsystems.

Note about notation: a_k^i indicates the area of item k while a_k^s indicates the area of shelf k. I is the set of items and S is the set of shelves. For any item x, p_x and s_x represent the item's probability and subsystem respectively.

The model is subject to the following constraints:

1. $\sum_{j=1}^{m} w_{ij} a_j^i \leq w_i^s, 1 \leq i \leq n.$
2. $\sum_{j=1}^{n} w_{ji} = 1, 1 \leq i \leq m.$
3. $\sum_{j=1}^{l} c_{ji} = 1, 1 \leq i \leq k.$
4. Let K be the set of indices of items belonging to subsystem i. Let Let L be the set of indices of shelves belonging to connected component j. Then $\sum_{a \in K} \sum_{b \in L} w_{ba} = |K| \cdot c_{ji}, 1 \leq i \leq k, 1 \leq j \leq l.$

Since each probability p is less than 1, we incentivize putting items with higher probability in the front by multiplying the probability by the distance. Thus, our objective function becomes

$$\min z = \sum_{i=1}^{n} \sum_{j=1}^{m} d_j p_i w_{ij}$$

Explanation of Constraints:

1. The sum of all of the items on a given shelf must be less than or equal to the maximum area that the shelf can support. We multiply by w_{ji} so that items not included on the shelf are not counted.

2. Each column of W can only have one 1, because we cannot put an item on two shelves simultaneously. Since each row represents a shelf, we simply traverse the whole column and ensure its sum is exactly 1. Notice that if we had put ≤ 1, the constraint would have been invalid because then no items would have been included. Thus, this constraint implicitly enforces the constraint that every item be included.

3. Each column of C can only have one 1 because all items belonging to a subsystem must be in the same connected component, and therefore a subsystem can be in exactly one connected component. Similarly to constraint 2, summing this and ensuring that the sum is 1 enforces the idea that every subsystem is included.

4. The intuition behind this constraint is that either all items belonging to a subsystem are in the same connected component, or none are. Thus, we use the decision variable c_{ji} to choose whether the sum is 0 or not.